rdrand Documentation

Release 1.49

Christopher Stillson

Dec 09, 2017

Contents:

CHAPTER 1

What is rdrand?

RdRand is a Python module that provides an easily used interface to the random number generator provided by Intel (and some Amd) processors. The **rdrand** and **rdseed** assembly instructions are used to generated random numbers, which are then fed into a subclass of the standard Random class of the random module. This provides a cryptographically strong random number generator with a familiar interface. Code that uses `random.Random` can be easily replace with a much stronger random number generator.

1.1 RdRandom

`rdrand.RdRandom` is a class that works exactly like `random.Random` See Python2_Random or Python3_Random for a description of `random.Random`

```
import rdrand

r = rdrand.RdRandom()

r.randint(10) ...
```

`RdRandom` uses the **rdrand** assembly instruction as the basis for its random numbers.

The only differences between `RdRandom` and `Random` are the following functions:

seed() and jumpahead() always returns None

getstate() and setstate() raises an exception

getrandbytes(num_bytes) Returns a string/bytes byte_count long of random bytes.

1.2 RdSeedom

RdSeedom work just like RdRandom, but uses the **rdseed** instruction.

CHAPTER 2

Special Methods and Variables

rdrand.HAS_RAND This variable is 1 if the CPU supports the **rdrand** instruction, 0 if it doesn't.

rdrand.HAS_SEED This variable is 1 if the CPU supports the **rdseed** instruction, 0 if it doesn't.

rdrand.rdrand_get_bits(bit_count) Returns a long with bit_count bits of randomness. Uses the **rdrand** instruction.

rdrand.rdrand_get_bytes(byte_count) Returns a string/bytes byte_count long of random bytes. Uses the **rdseed** instruction.

rdrand.rdseed_get_bits(bit_count) Returns a long with bit_count bits of randomness. Uses the **rdseed** instruction.

rdrand.rdseed_get_bytes(byte_count) Returns a string/bytes byte_count long of random bytes. Uses the **rdseed** instruction.

CHAPTER 3

Cryptographic Use

For general use and keys with a relatively short lifetime, `RdRandom` or `rdrand.rdrand_get_byte()` will provide a decent amount of entropy.

```
import rdrand

r = rdrand.RdRandom()

key = r.getrandbytes(16)
```

This will generate a 128 bit key.

For longer keys that may need to last longer (years, decades), you can use `RdSeedom`

```
import rdrand

s = rdrand.RdSeedom()

key = s.getrandbytes(32)
```

This will produce a 256 bit key, which should be completely unrelated to any previous or future keys.

CHAPTER 4

Warnings

I am not a cryptographer.

RdRand can be subverted by an attacker who controls a machine.

Use at your own risk.

Also, RdRand is not as fast as default random number generator. It is faster than other cryptographically secure random number generators, but if you don't need cryptographically secure random numbers it's probably overkill.

CHAPTER 5

References

https://software.intel.com/en-us/blogs/2012/11/17/the-difference-between-rdrand-and-rdseed

https://en.wikipedia.org/wiki/RdRand

https://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R__DRNG_Software_Implementation_Guide_final_Aug7.pdf

https://www.cryptopp.com/wiki/RDRAND_and_RDSEED

http://cr.yp.to/talks/2014.05.16/slides-dan+tanja-20140516-4x3.pdf

CHAPTER 6

Indices and tables

- genindex
- modindex
- search